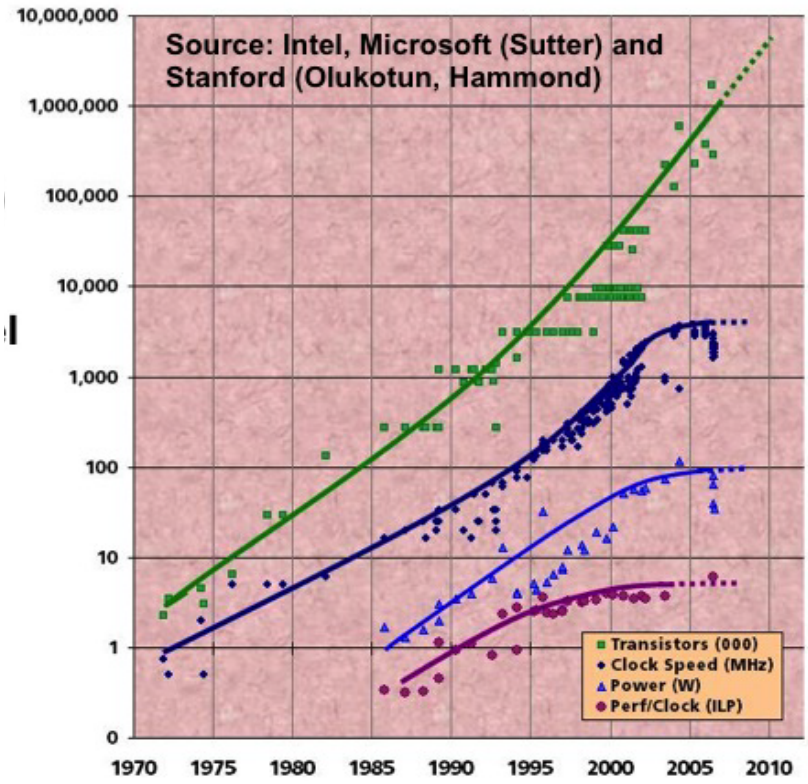
EDGS 6304 Computer Architecture

Ivor Page, The University of Texas at Dallas

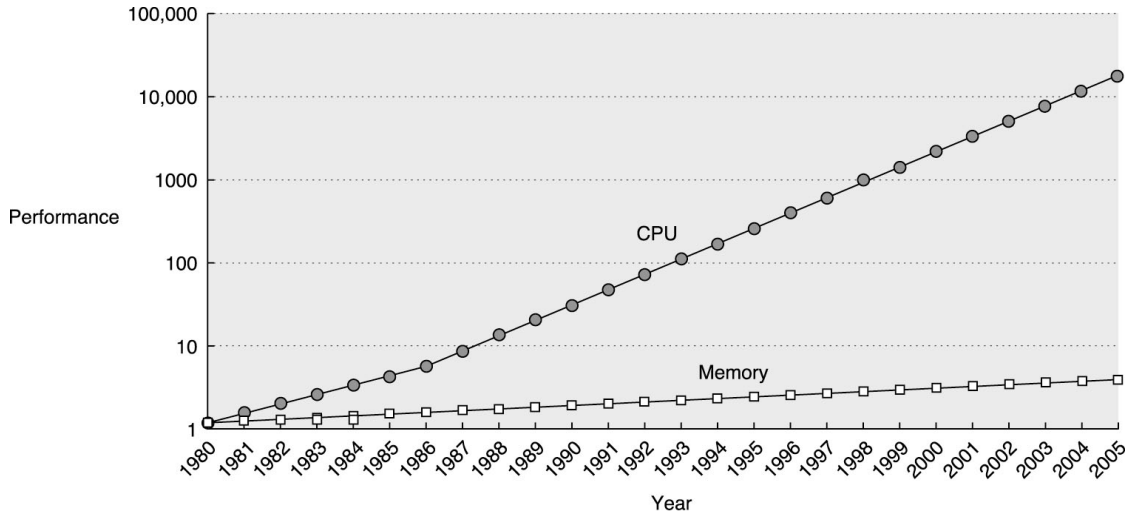
Lecture #2, Memory

1 Overall Performance



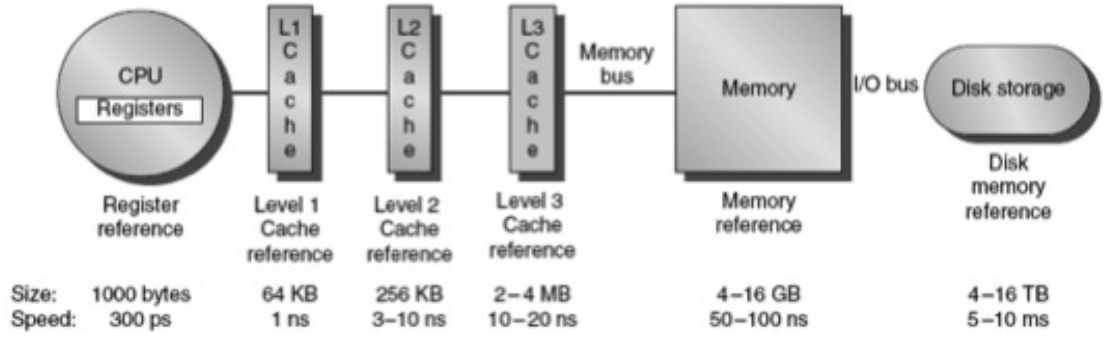
Chip density increases by 2x every 2 years.
Clock speed is increasing very slowly.
ILP is not increasing.
Conclusion: parallelism - multi-core chips.

1.1 Processor - Memory Speed Gap



Conclusion: memory hierarchy is essential: high speed, small L1 caches, larger, slower L2 caches, . . ., huge slow RAM, Flash, Disk, etc.

1.2 Memory Hierarchy



Example: Nehalem Quad Core
For each core:
L1: 32K instruction and 32K data cache,
L2: 1MB
L3: 8MB shared among all 4 cores

2 Cache Memories

There are three cache memory organizations, Direct Mapped, Fully Associative, and Set Associative. In each case the cache memory holds *Lines* or *Blocks* of words of memory. Typically a block is 64 bytes, but block size is one of the factors that most influences *Hit Rate* - the fraction of accesses to memory that find their target data in the cache.

A memory read first goes to the L1 cache. If a hit occurs, the block containing the word, double word, half word, or byte needed is fetched and the required component selected by multiplexers. Subsequent accesses to elements within that block can be faster than the first if the cache or processor includes a block register.

The total size of the cache also greatly influences hit rate.

2.1 Cache Memories: Principle of Locality

In most modern architectures there are two L1 caches, one for instructions and one for data, not necessarily of the same size.

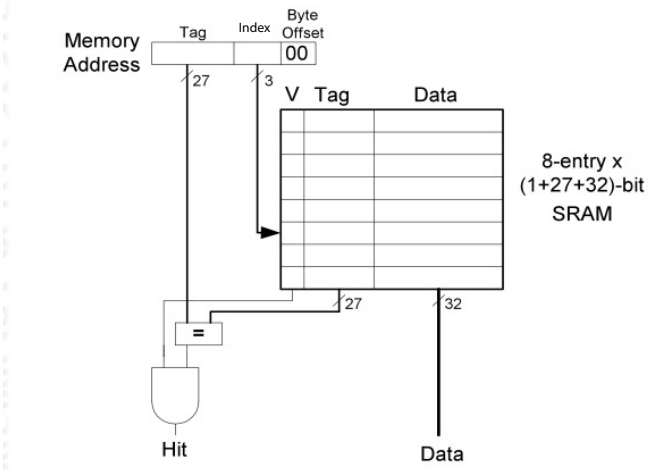
Instruction caches are not written-to by user programs or the operating system.

Accesses to instruction caches benefit more from the *Principle of Locality* than do data caches. In particular, accesses are more sequential in the instruction cache.

Temporal Locality: If an item is accessed, it will probably be accessed again in the near future (Loops of instructions).

Spacial Locality: Items close by to an item that has just been accessed are likely to be accessed soon (Indexing through a vector or an array).

2.2 Direct Mapped Caches, Simple 8-block cache



2.3 1024 block Direct Mapped Cache with 64 byte blocks

16	10	4	2
Tag	Index	Word	Byte

A 32 bit address split into its components for a 1024 block cache with 64 bytes per block.

V	Tag	Data Block

1024 entries

V = Valid bit.

2.4 Addressing: Direct Mapped

Consider RAM to be partitioned into blocks of 64 bytes with start addresses 0, 64, 128, 192, etc. Call these Block 0, Block 1, Block 2, etc.

RAM Block i corresponds with cache block i mod 1024.

Blocks 0, 1024, 2048, 3072 of RAM all map to block zero of the cache.

When a memory reference is presented to the cache, the index field gives the cache block number. The Tag field of the memory address is compared with the Tag value stored in the cache. If there is a match, a hit has occurred and the read or write process continues, working on the required word/byte of the data block for that index value.

If a miss occurs, then the required block must be fetched from the next level of cache memory. A *Write Back* may be necessary before the *old* cache block can be overwritten.

There is no freedom as to where a block of RAM must be placed in the cache. The direct mapped cache organization suffers from **aliasing**, meaning that multiple RAM addresses map to the same location in the cache.

2.5 Fully Associative Organization

26	4	2
Tag	Word	Byte

A 32 bit address split into its components for a 1024 block cache with 64 bytes per block.

V	Tag	Data Block

1024 entries

Now the entire 26 bit memory address Tag field must be compared with all 1024 Tag fields in the cache entries.

To enable a 1024-block fully associative cache memory, we would need 1024 26-bit comparators.

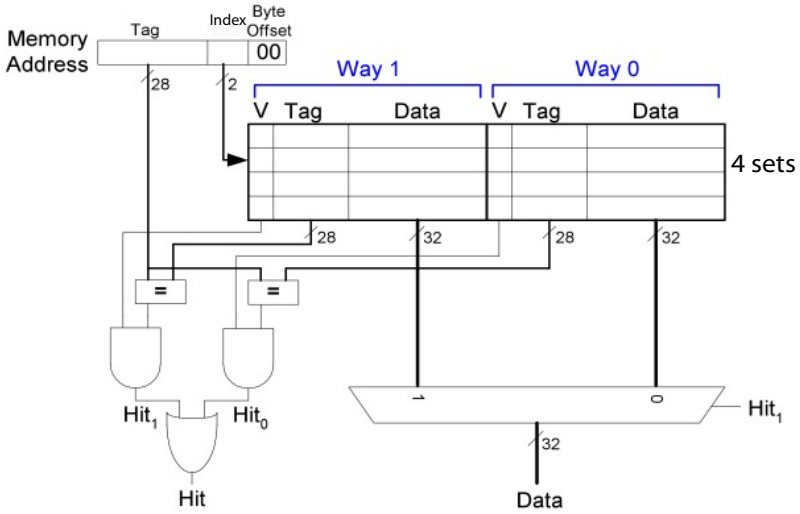
Fully associative design enables any block of RAM to reside in any block of the cache, which tends to minimize conflict misses.

The fully associative design is extremely expensive in hardware and may be slower than other designs. It is largely of academic interest in finding the best possible performance with a given block size and a number of cache blocks.

2.6 Set Associative Organization

Imagine a direct mapped cache in which there are two, four, or eight blocks in the cache associated with an index value.

The memory address maps to an index in the cache memory. Then an associative match takes place between the tag in the memory address and the tags stored in the cache for that index.



Simple 2-way set associative cache with one word of 32 bits per block and 4 sets.

2.6.1 2-way set associative organization, 512 sets, blocks of size 64Bytes.

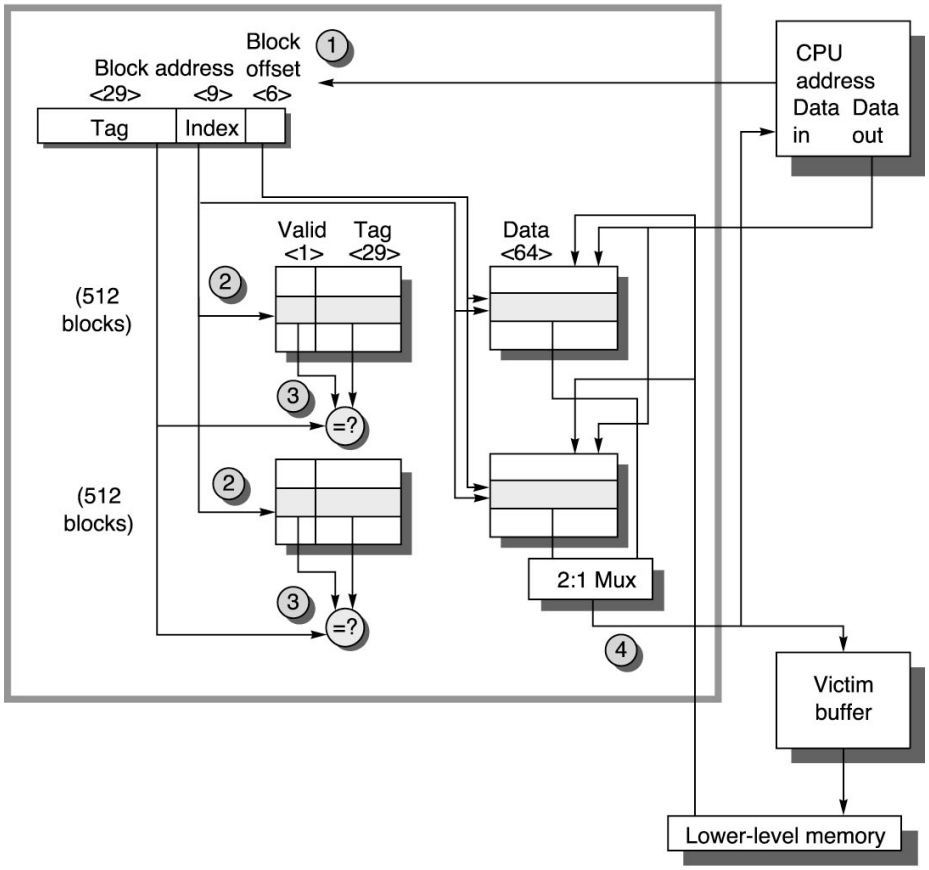
17	9	4	2
Tag	Index	Word	Byte

A 32 bit address split into its components for a two-way set associative cache with 512 sets and 64 bytes per block.

V	Tag	Data Block

512 sets

2-Way Set Associative Opteron Cache, 64 KB, 64 Byte Blocks, 512 sets



The memory address index field accesses one of the 512 sets in the cache table. Then the memory address tag field is compared in parallel with the two tag fields stored in that set of the cache. If there is a hit in one of the comparisons, the cache access continues as normal.

Otherwise a miss has occurred and one of the two cache blocks of that set must be replaced or overwritten.

The comparison logic is now simpler since it only has to compare the memory address tag field (of 17 bits) with two possible cache tags.

2.7 Number of Bits in the Cache Table

If we stay with the 1024 block cache, where each block is 64 bytes, we get the following numbers of bits for different organizetions:

Organization	Valid Bits	Tag Bits	Data Bits	Size, kBytes
Direct Mapped	1*1024	16*1024	64*8*1024	67.712
Fully Associative	1*1024	26*1024	64*8*1024	68.992
2-Way Set Associative	1*1024	17*1024	64*8*1024	67.84
4-Way Set Associative	1*1024	18*1024	64*8*1024	67.968
8-Way Set Associative	1*1024	19*1024	64*8*1024	68.096

Note that the data part is 65.536 kBytes. The overhead for each of the above designs is no more than 5.3%.

2.8 Characterizing Cache Misses

Cache missed occur because of three reasons:

1. **Compulsory**
   Also called Cold Start Misses or First Reference Misses. The first access to a block from RAM causes a compulsory miss. These misses would have occurred in an infinite sized cache.

2. **Capacity**
   If the cache is too small to hold all of the blocks needed during execution of a program, misses occur on blocks that were discarded earlier. The capacity miss rate of ANY cache is defined as the miss rate of an *ideal* cache of the same size minus the compulsory miss rate. i.e. it is independent of the organizational details of the cache being considered, and only depends on its size.

   The ideal cache mentioned here must only suffer from compulsory and capacity misses. A fully associative cache with ANY replacement policy is NOT ideal because an access sequence can always be chosen that causes more than the absolute minimum number of misses. (see the examples that follow).

3. **Conflict**
   The remaining misses are conflict misses.

   If the cache has sufficient space for the data, but the block cannot be kept because the set is full, a conflict miss will occur. The conflict miss rate is defined as the difference between the miss rate of the cache being considered and an ideal cache of the same size. These misses are also called collision or interference misses.

2.8.1 Example of the 3Cs

Consider a cache that can hold 4 blocks. A program repeatedly traverses an array of 5 blocks. Here is the address pattern (A,B,C,D,E)*.

In all caches, the first accesses to the five blocks cause compulsory misses.

In a direct mapped cache of 4 blocks, blocks A and E compete for the first block of the cache. The cache contents cycle between A B C D and E B C D. There are 2 misses per iteration, following the first.

An ideal cache generates the minimum possible numbers of replacements. To achieve the ideal performance with this address sequence, we could use a fully associative cache with the Most Recently Used Replacement Scheme. It would generate 2 misses per iteration. Therefore, since the ideal cache cannot generate conflict misses, and the misses in both the fully associative cache using MRU and the direct mapped cache are the same (2 per iteration) the misses in both caches must be capacity misses.

Now consider a fully associative cache using LRU.

The contents would progress as follows:

Accessed	Contents			
A	A	-	-	-
B	A	B	-	-
C	A	B	C	-
D	A	B	C	D
E	E	B	C	D
A	E	A	C	D
B	E	A	B	D
C	E	A	B	C
D	D	A	B	C
E	A	E	B	C

Each fetch of a block replaces the block needed next, leading to 5 misses per cycle.

Clearly this is not ideal operation. The ideal cache generates only two misses per cycle. Three of the misses in the fully associative cache using LRU must be conflict misses.

Although a fully associative cache using MRU replacements would be ideal for the above

address sequence, let's consider the same cache with the address sequence A B C D (E D)* repeated indefinitely. Here is sequence of cache contents:

Accessed	Contents
A	A - - -
B	A B - -
C	A B C -
D	A B C D
E	A B C E
D	A B C D
E	A B C E
D	A B C D
E	A B C E

After the cache becomes full the algorithm iterates between just two blocks of the array. Each iteration over the two blocks E, D, causes two misses.

With this address sequence, a 4-block associative cache using LRU would have the following contents.

Accessed	Contents			
A	A	-	-	-
B	A	B	-	-
C	A	B	C	-
D	A	B	C	D
E	E	B	C	D
D	E	B	C	D
E	E	B	C	D
D	E	B	C	D
E	E	B	C	D

No misses after D and E are loaded into the cache.

For completion, consider a 2-way set associative cache with two sets of two blocks each. Blocks A, C, E compete for the first set and B and D compete for the second set. With the LRU replacement scheme operating in each set, and the access sequence (A B C D E)*, the contents of the cache would progress as follows:

Accessed	Contents			
	Set 0		Set 1	
A	A	-	-	-
B	A	-	B	-
C	A	C	B	-
D	A	C	B	D
E	E	C	B	D
A	E	A	B	D
B	E	A	B	D
C	C	A	B	D
D	C	A	B	D
E	C	E	B	D
A	A	E	B	D
B	A	E	B	D
C	A	C	B	D
D	A	C	B	D

That's 3 misses per cycle, two capacity misses and one conflict miss.

2.8.2 The Ideal Cache

The ideal cache would keep a block of RAM X in the cache while the number of accesses to distinct RAM blocks other than X, since X was last accessed, was not larger than the size of the cache.

The cache would have to be fully associative so that blocks were not replaced because of address aliasing. It is an impractical idea, except in evaluating real caches.

In my program for simulating an arbitrary cache, I maintained, for each block of RAM, X, a set that contained all of the RAM block numbers that had been accessed since X was last accessed. When a miss occurred and the cache under test replaced RAM block Y, I checked the size of Y's set. If it equalled or exceeded the cache size then I recorded a capacity miss, otherwise I recorded a conflict miss.

On each access to a RAM block I cleared that block's set.

2.9 Effects of Cache Size

Percentage Miss Rate			
Size	Inst'n Cache	Data Cache	Unified Cache
1 KB	3.06	24.61	13.34
2 KB	2.26	20.57	9.78
4 KB	1.78	15.94	7.24
8 KB	1.10	10.19	4.57
16 KB	0.64	6.47	2.87
32 KB	0.39	4.82	1.99
64 KB	0.15	3.77	1.35
128 KB	0.02	2.88	0.95

These data are for a direct mapped cache with 32 Byte blocks for an average of SPEC92 benchmarks. The percentage of instructions in the mix is about 75%

2.10 Effects of Cache Size, Associativity

Cache Size	Degree Associativity	Total Miss Rate	Miss Rate Components (relative Percent) Compulsory		Capacity		Conflict	
2 KB	1-way	0.098	0.002	2%	0.044	45%	0.052	53%
2 KB	2-way	0.076	0.002	2%	0.044	58%	0.030	39%
2 KB	4-way	0.064	0.002	3%	0.044	69%	0.018	28%
2 KB	8-way	0.054	0.002	4%	0.044	82%	0.008	14%
4 KB	1-way	0.072	0.002	3%	0.031	43%	0.039	54%
4 KB	2-way	0.057	0.002	3%	0.031	55%	0.024	42%
4 KB	4-way	0.049	0.002	4%	0.031	64%	0.016	32%
4 KB	8-way	0.039	0.002	5%	0.031	80%	0.006	15%
8 KB	1-way	0.046	0.002	4%	0.023	51%	0.021	45%
8 KB	2-way	0.038	0.002	5%	0.023	61%	0.013	34%
8 KB	4-way	0.035	0.002	5%	0.023	66%	0.010	28%
8 KB	8-way	0.029	0.002	6%	0.023	79%	0.004	15%
16 KB	1-way	0.029	0.002	7%	0.015	52%	0.012	42%
16 KB	2-way	0.022	0.002	9%	0.015	68%	0.005	23%
16 KB	4-way	0.020	0.002	10%	0.015	74%	0.003	17%
16 KB	8-way	0.018	0.002	10%	0.015	80%	0.002	9%
32 KB	1-way	0.020	0.002	10%	0.010	52%	0.008	38%
32 KB	2-way	0.014	0.002	14%	0.010	74%	0.002	12%
32 KB	4-way	0.013	0.002	15%	0.010	79%	0.001	6%
32 KB	8-way	0.013	0.002	15%	0.010	81%	0.001	4%
64 KB	1-way	0.014	0.002	14%	0.007	50%	0.005	36%
64 KB	2-way	0.010	0.002	20%	0.007	70%	0.001	10%
64 KB	4-way	0.009	0.002	21%	0.007	75%	0.000	3%
64 KB	8-way	0.009	0.002	22%	0.007	78%	0.000	0%
128 KB	1-way	0.010	0.002	20%	0.004	40%	0.004	40%
128 KB	2-way	0.007	0.002	29%	0.004	58%	0.001	14%
128 KB	4-way	0.006	0.002	31%	0.004	61%	0.001	8%
128 KB	8-way	0.006	0.002	31%	0.004	62%	0.000	7%

2.11 Effects of Cache Size, Associativity

Compulsory Misses do not vary with cache size and associativity.

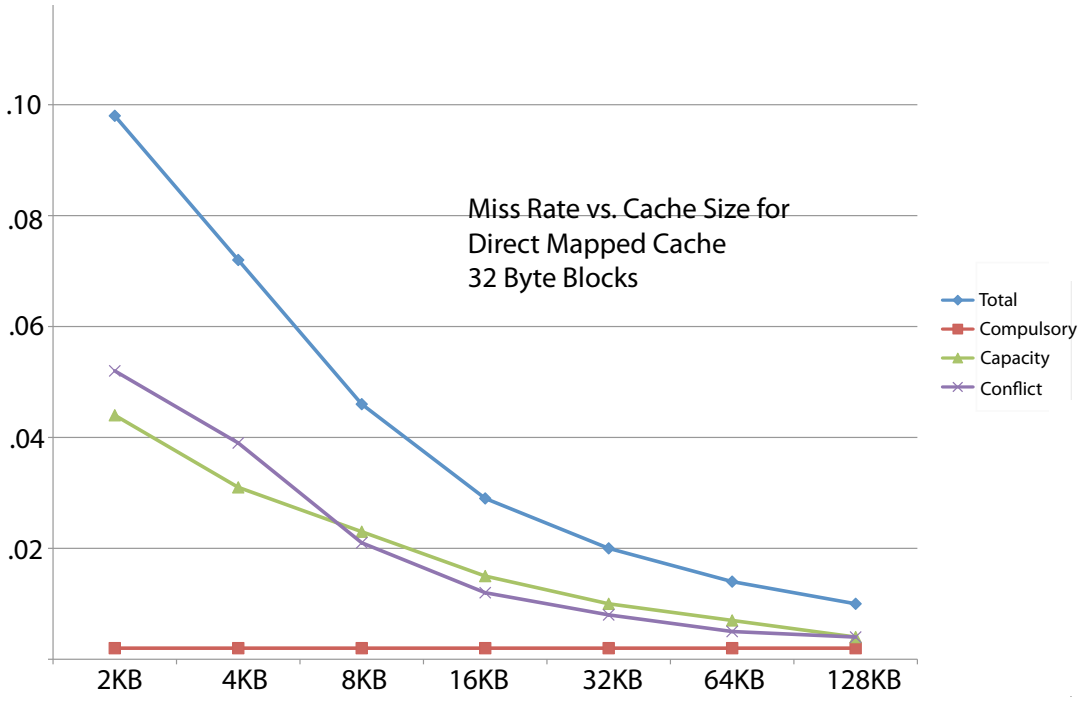
Capacity Misses reduce with Cache Size, but not with Associativity.

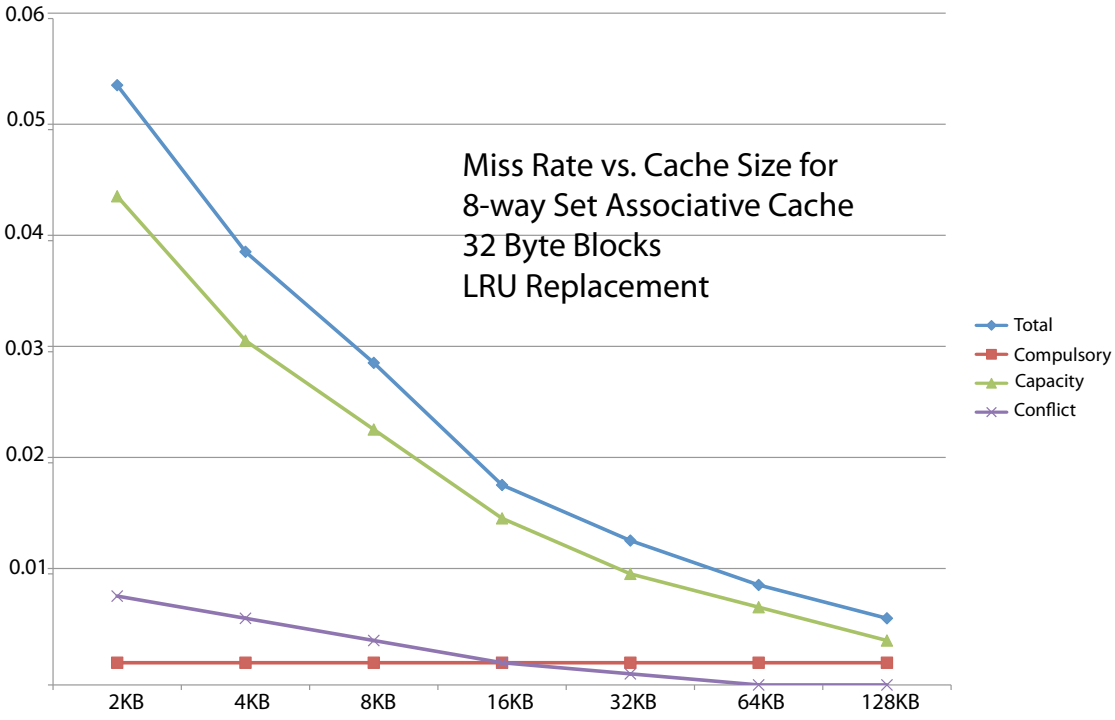
Conflict Misses reduce with both Cache Size and Associativity.

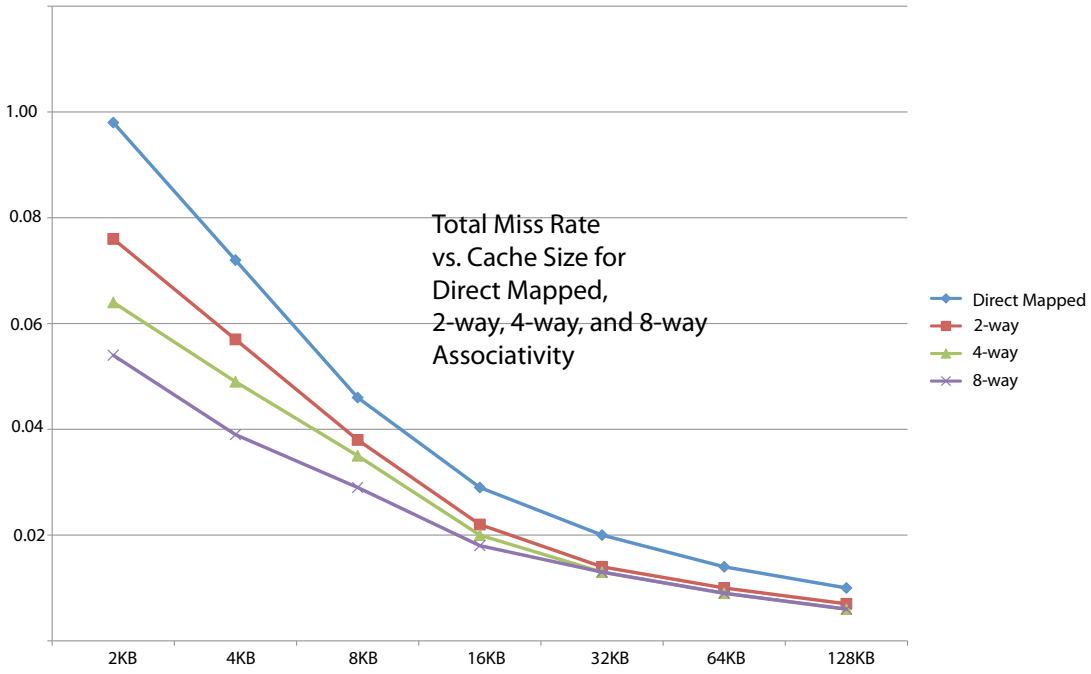
Almost all misses are due to Capacity as Associativity increase.

Total Miss Rate varies little between Associativity 2, 4, and 8 and size above 4kB.

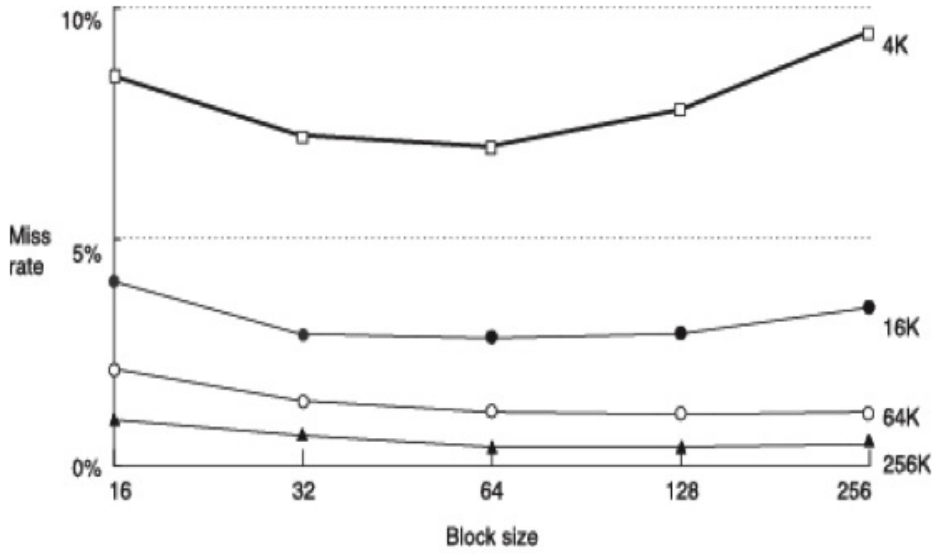
A Direct mapped cache of size N has about the same miss rate as a 2-way set associative cache of size N/2.


.10
.08
.06
.04
.02
Miss Rate vs. Cache Size for
Direct Mapped Cache
32 Byte Blocks
Total
Compulsory
Capacity
Conflict
2KB
4KB
8KB
16KB
32KB
64KB
128KB


0.06
0.05
0.04
0.03
0.02
0.01
Miss Rate vs. Cache Size for
8-way Set Associative Cache
32 Byte Blocks
LRU Replacement
Total
Compulsory
Capacity
Conflict
2KB
4KB
8KB
16KB
32KB
64KB
128KB

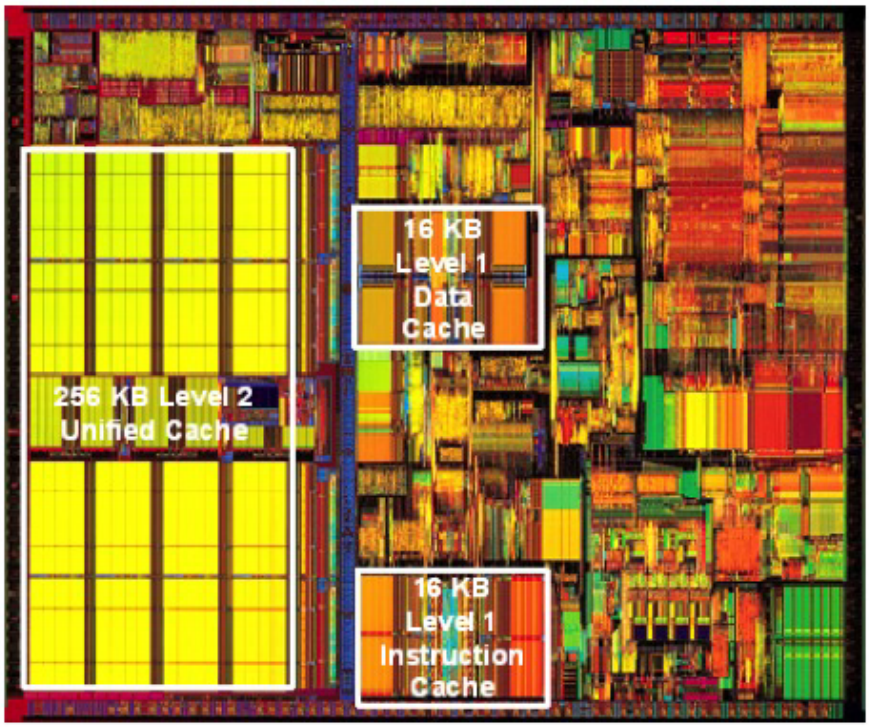


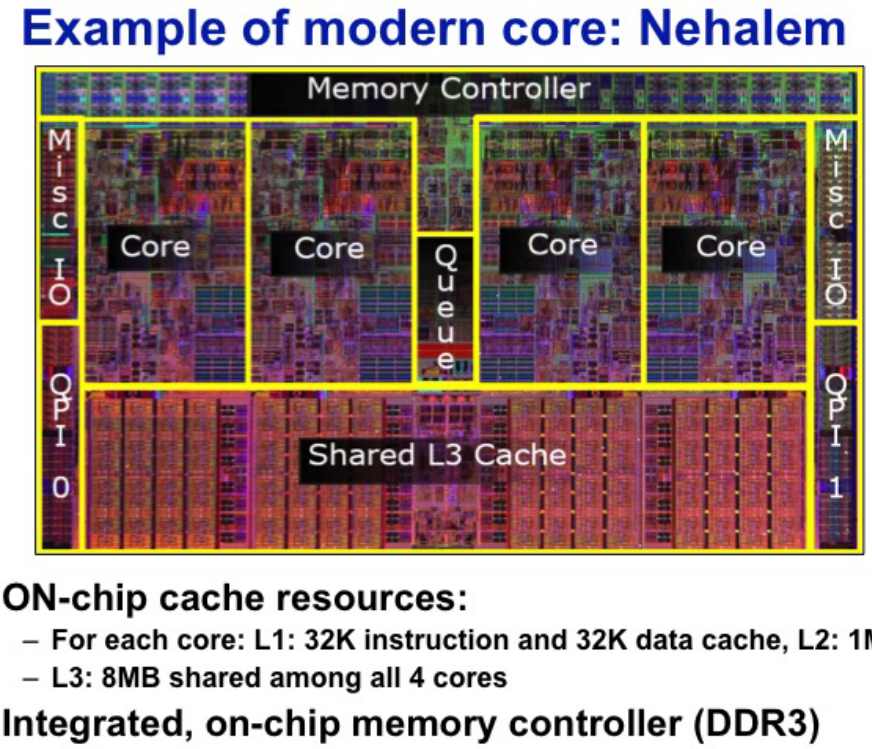
2.12 Effects of Block Size



- **Bigger blocks reduce compulsory misses**
- **Bigger blocks increase conflict misses**

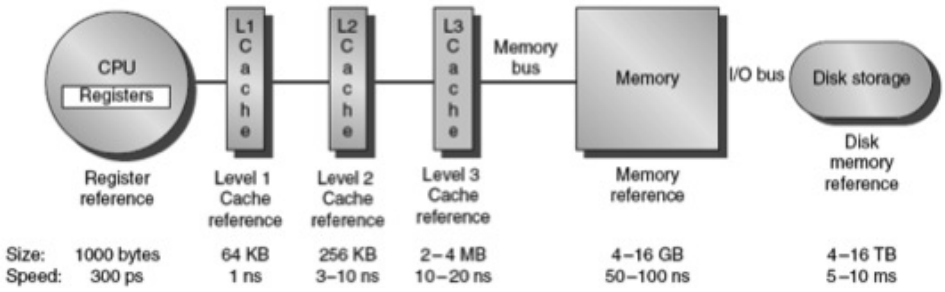
2.13 Pentium III





- **ON-chip cache resources:**
  - **For each core: L1: 32K instruction and 32K data cache, L2: 1MB**
  - **L3: 8MB shared among all 4 cores**
- **Integrated, on-chip memory controller (DDR3)**

2.14 Memory Hierarchy



2.15 Access Time Calculations, Unified L1 Cache

Consider an L1 cache with the following miss rates and miss penalties in cycles in a processor where the CPI is one cycle for each cache hit:

		Miss Rate %			
		1	2	5	10
Penalty	1	1.01	1.02	1.05	1.10
	2	1.02	1.04	1.10	1.20
	5	1.05	1.10	**1.25**	1.5
	10	1.10	1.20	1.50	2.0

With a 5% miss rate and a miss penalty of 5 cycles, the CPI increases from 1.0 to 1.25.

The average access time for a memory system with a unified cache is

$$\text{Average access time on hit} + \text{Miss penalty} \times \text{Miss rate}$$

2.16 Access Time Calculations, Multilevel Cache

Consider the following access times in processor clock cycles and miss-rates:

Level	L1	L2	L3	RAM
Access Time	1	5	20	100
Miss Rate	5%	10%	2%	0%

$$\begin{aligned}
\text{Average access time} &= \text{Access time on hit}_{L1} + \text{Miss rate}_{L1} \times \text{Miss penalty}_{L1} \\
\text{Miss penalty}_{L1} &= \text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2} \\
\text{Miss penalty}_{L2} &= \text{Hit time}_{L3} + \text{Miss rate}_{L3} \times \text{Miss penalty}_{L3}
\end{aligned}$$

$$\begin{aligned}
\text{Average access time} &= 1 + (5/100)[5 + (10/100)[20 + (2/100)[100]]] \\
&= 1 + (5/100)5 + (5/100)(10/100)20 + (5/100)(10/100)(2/100)100 \\
&= 1 + 0.25 + 0.1 + 0.01 = 1.36
\end{aligned}$$

a 36% increase in CPI, or a 36% loss in performance.

Most of the loss in due to L1 miss rate and L2 speed.

2.17 Access Time Calculations, Split L1 Cache

The CPI will be affected independently by Icache and Dcache misses:

CPI = (Cycles for Icache hit) +
(Icache miss rate)×(penalty for Icache miss) +

(fraction memory reference instructions)×(cycles for Dcache hit +
(Dcache miss rate)×(Dcache miss penalty))

Size(KB)	Instruction Cache	Data Cache	Unified Cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.002	32.6	32.9

Misses per 1000 instructions. 74% of all cache references are for instructions, 26% for data. The data are for a 2-way set associative cache with 64 byte blocks.

Which has the lower miss rate, a 16 KB instruction cache plus a 16 KB data cache, or a 32KB single-ported unified cache?

Assume 36% of instructions are loads or stores. Why? out of every 100 cache accesses, 74 will be for instructions and 26 will be for data. Therefore $26/74 = 35.13\%$ of the instructions will be loads and stores. The question "rounds this figure up" to 36%.

Assume a hit takes 1 cycle and a miss takes 200 cycles. Assume that a load or a store instruction on a unified cache takes 2 cycles since the cache is single-ported. Assume also that a load or store instruction on a split cache takes one cycle.

The execution model here, which assumes overlapped accesses for an instruction and a data access, must be of a pipelined architecture. In MIPS, for example, the IF stage fetches the next instruction. Then comes ID, EX, and then the Mem, the Memory Access Stage. Finally comes WB. If we have a unified L1 cache then, when a load/store instruction is in the Mem stage, the IF stage cannot fetch an instruction. The load/store instructions causes a one-cycle stall (with a unified L1 cache) and in this question we charge that cycle to the cache access for data.

Converting misses per 1000 instructions to miss rate, and using the figures from the table for a 16kB instruction cache, a 16kB data cache, and a 32kB unified cache,

$$\text{The instruction cache miss rate} = \frac{\frac{\text{Misses during 1000 instructions}}{1000}}{\text{Memory accesses per instruction}}$$

There is one access to the instruction cache for each instruction. Therefore,

$$\text{The instruction cache miss rate} = \frac{3.82/1000}{1.0} \approx 0.004$$

There are 0.36 accesses to the data cache per instruction. Therefore,

$$\text{The data cache miss rate} = \frac{40.9/1000}{0.36} \approx 0.114$$

There are 1.00 + 0.36 accesses to the unified cache per instruction. Therefore,

$$\text{The unified cache miss rate} = \frac{43.3/1000}{1.00 + 0.36} \approx 0.0318$$

Combining the two miss rates for the split cache, noting that 75% of cache accesses are for the instruction cache and 26% are for the data cache,

$$\text{The overall miss rate for the split cache} = (75\% \times 0.004) + (26\% \times 0.114) = 0.0326$$

This is the sum of the probabilities for each event times the miss rate for that event. We are assuming that the events are independent.

The unified cache has a slightly lower miss rate than the split cache.

So far we just have miss rates. Now we must compute average access times. We will use clock cycles in place of times, since we don't know actual times in picoseconds or nanoseconds.

We use the standard formula,

$$\text{Average access time for a cache} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

The average access time is the sum of the average access time for instructions plus the average access time for data. We sum the probabilities of each event times the time duration of that event.

AverageAccessTime =
Fraction of accesses for inst'ns × (Icache Hit Time + Icache Miss Rate × Icache Miss Penalty) +
Fraction of accesses for data × (Dcache Hit Time + Dcache Miss Rate × Dcache Miss Penalty)

Average Access Time for the Split Cache =
$0.74 \times (1 + 0.004 \times 200) + 0.26 \times (1 + 0.114 \times 200) = 7.52$

Average Access Time for the Unified Cache =
$0.74 \times (1 + 0.0318 \times 200) + 0.26 \times (2 + 0.0318 \times 200) = 7.62$

Notice that the unified cache has a hit time of 2 clock cycles for data accesses, as explained earlier.

Another way to think about this - Cycles per Instruction, CPI
In every 100 instructions, 36 will be loads and stores and 64 will be inter-register instructions. All 100 will require an access to the instruction cache, while the 36 loads and stores will require an additional access to the data cache.

For the split cache system, the average CPI will be:

Hit Time in Icache + miss rate in Icache × miss penalty + fraction of instn's that are loads/stores × (Hit time in data cache + miss rate in Dcache × miss penalty in Dcache) =

$1 + 0.004 \times 200 + 0.36 \times (1 + 0.114 \times 200) = 1 + 0.8 + 8.57 = 10.37$

For the unified cache, the Icache and the Dcache have the same miss rates. The CPI will be

$1 + 0.0318 \times 200 + 0.36 \times (2 + 0.0318 \times 200) = 1 + 6.36 + 2.65 = 10.36$

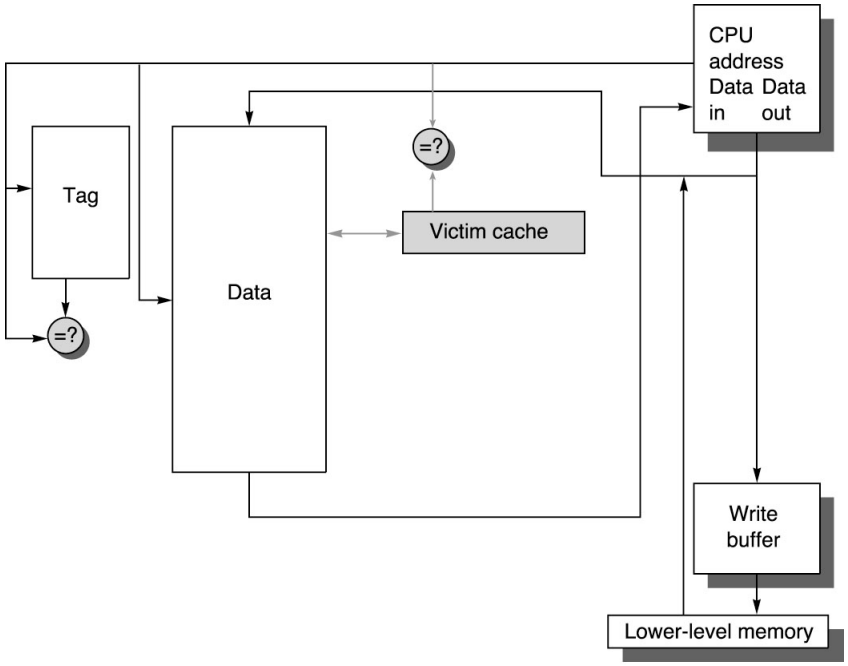
2.18 Miss Policy

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

2.19 Write Policy

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

2.20 Write Buffer



The write buffer contains a small number of data blocks that are waiting to be written to the next lower level. It uses fully associative addressing. All L1 misses are checked against the write buffer. Blocks can be reclaimed from the write buffer to the L1 cache.

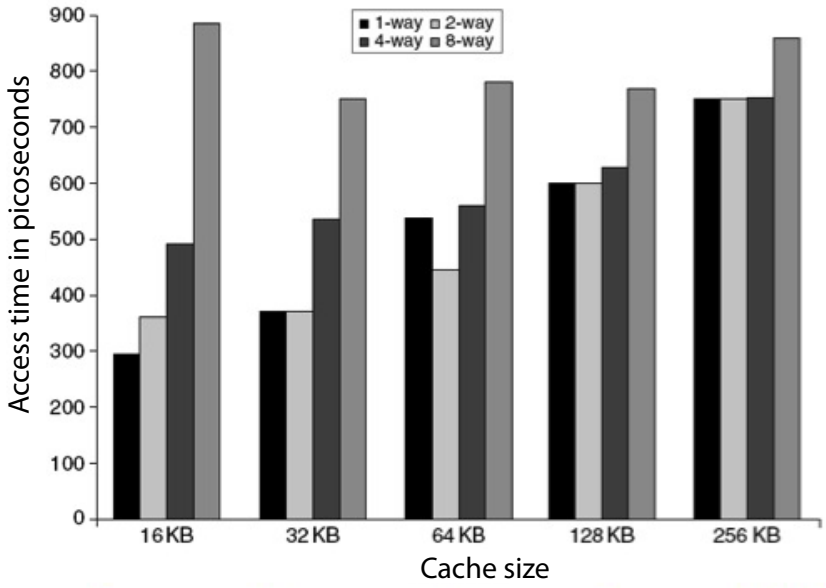
2.21 Six Cache Optimizations

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

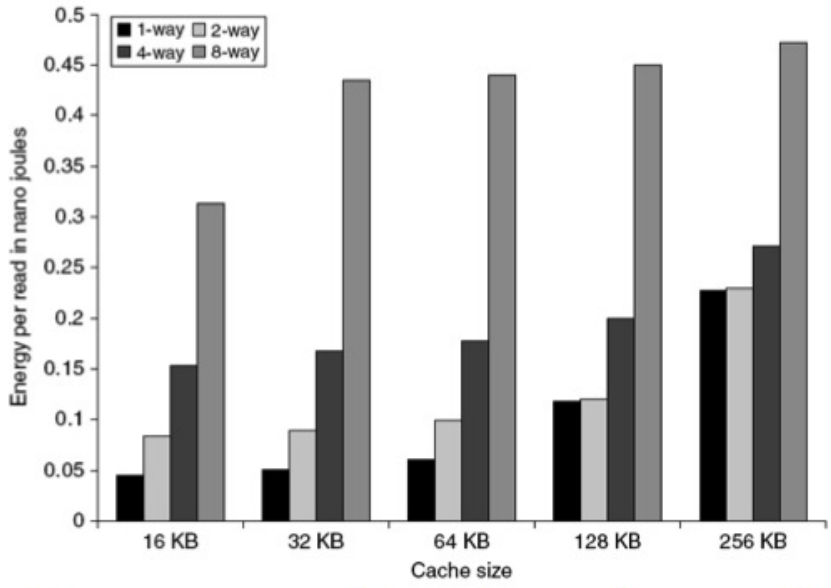
2.22 Ten Advanced Optimizations

- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

2.23 L1 Size and Associativity



Access time vs. size and associativity



Energy per read vs. size and associativity

2.24 2: Way Prediction

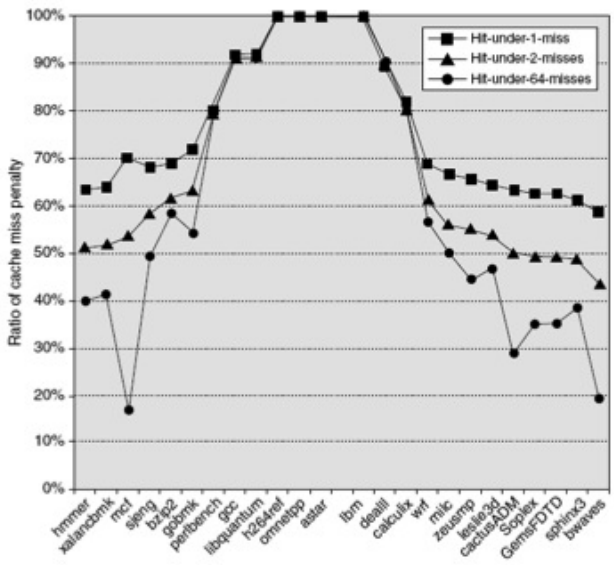
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - "Way selection"
  - Increases mis-prediction penalty

2.25 3: Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

2.26 4: Non-Blocking Cache

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



2.27 5: Multi-Bank Cache

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address

Block address	Bank 0	Block address	Bank 1	Block address	Bank 2	Block address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

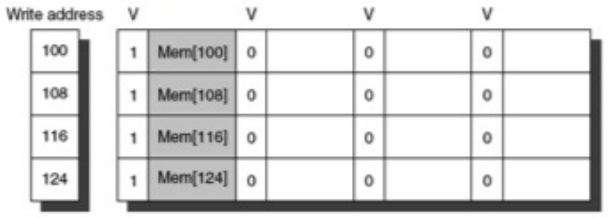
2.28 6: Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

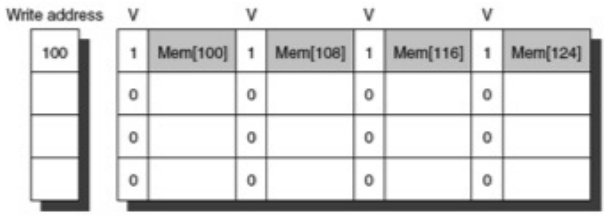
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

2.29 7: Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering



Write buffering

2.30 8: Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

2.31 9: Loop Interchange

```
/* Before */
   for(j=0; j<100; j++)
       for(i=0; i<5000; i++)
           x[i][j] = 2*x[i][j];

/* After */
   for(i=0; i<5000; i++)
       for(j=0; j<100; j++)
           x[i][j] = 2*x[i][j];
```

The "before" code skips through memory in strides of 100, while the "after" code access memory in sequence (stride = 1).

2.32 10: Blocking: Matrix Multiply

```
/* Before */
   for(i=0; i<N; i++)
       for(j=0; j<N; j++) {
           r = 0;
           for(k=0;k<N;k++)
               r = r + y[i][k]*z[k][j];
           x[i][j] = r;
       }
```

The code accesses one entire row of y and one entire column of z for every value of x computed. Cache misses depend on the relative sizes of N and the cache. If the cache is much smaller than $3 \times N^2$ then capacity misses and conflict misses will occur.

The total number of memory accesses is $2N^3 + N^2$.

2.33 8: Blocking: Matrix Multiply

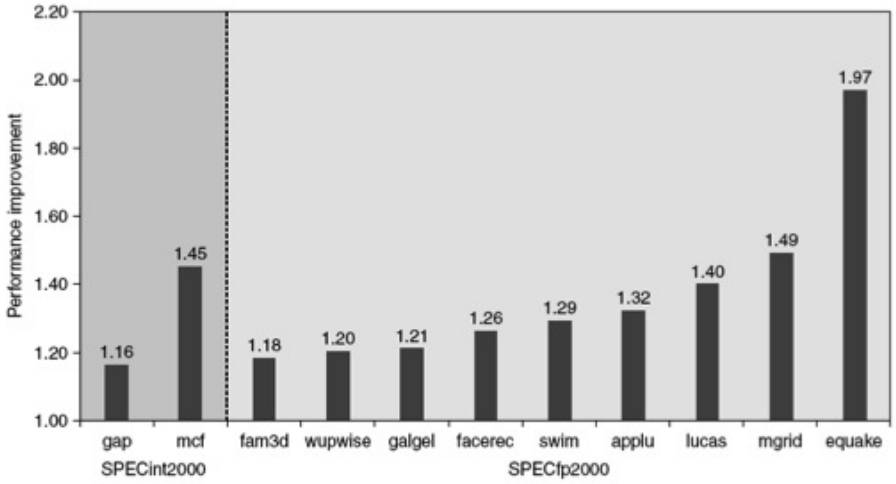
```
/* After */
   for(jj=0; jj<N; jj=jj+B)
       for(kk=0; kk<N; kk=kk+B)
           for(i=0; i<N; i++)
               for(j=jj; j<min(jj+B,N); j++) {
                   r = 0;
                   for(k=kk; k<min(kk+B;N);k++)
                       r = r + y[i][k]*z[k][j];
                   x[i][j] = x[i][j] + r;
               }
```

The inner two loops now access blocks of size B^2 in both y and z in each iteration of the i-loop. Each of these blocks is never accessed again.

The total number of memory accesses is $2N^3/B + N^2$, an improvement of rougly a factor of B.

2.34 9: Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Two SpecInt2000 at the left and nine Specfp2000 at the right.
Only those programs of SpecInt2000 that benefitted more than 15% are shown.

2.35 10: Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions
- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache
- Combine with loop unrolling and software pipelining

2.36 Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.